

UMesh^{2D} User Manual

S. Rebay, A. Guardone and D. Dussin

October 15, 2013

Abstract

This manual explains how to use program UMesh^{2D} to generate two dimensional unstructured isotropic grids of Delaunay type by means of the Delaunay advancing front method of Rebay [1]. A detailed description of the input files necessary to define and triangulate arbitrary computational regions in two dimensions is given. The program can deal with multiply connected domains and allows for specifying the local dimension of the triangles at various locations on its boundary and within the domain. A complete example is provided to illustrate the structure and format of the input data as well as the form of the output files.

Revised by N. Calosso and L. Quartapelle

Contents

1	Introduction	2
2	Overview of the program	2
2.1	Input files	2
2.2	Grid generation process	3
2.3	Output files	4
3	Input file for UMesh^{2D}	4
3.1	The file <code>boundary.grid_name</code>	5
3.2	The file <code>domain.grid_name</code>	10
4	Output files of UMesh^{2D}	11
4.1	Intermediate files: <code>knots.grid_name</code> and <code>geometry.grid_name</code> .	11
4.2	Files of the final mesh: <code>nodes.grid_name</code> and <code>grid.grid_name</code> .	11
4.3	Plotting files: <code>gridplot.grid_name</code> and <code>backplot.grid_name</code> . .	13
5	Messages and errors on using UMesh^{2D}	15
5.1	Echoes on the screen	15
5.2	Frequent errors	15

1 Introduction

UMesh^{2D} is a program for generating 2D unstructured grids of triangles. The generation produces a Delaunay triangulation [2] (see the example in Figure 1). The program implements the Delaunay advancing method developed by Stefano Rebay which is centered around an original and smart use of Bowyer–Watson algorithm.

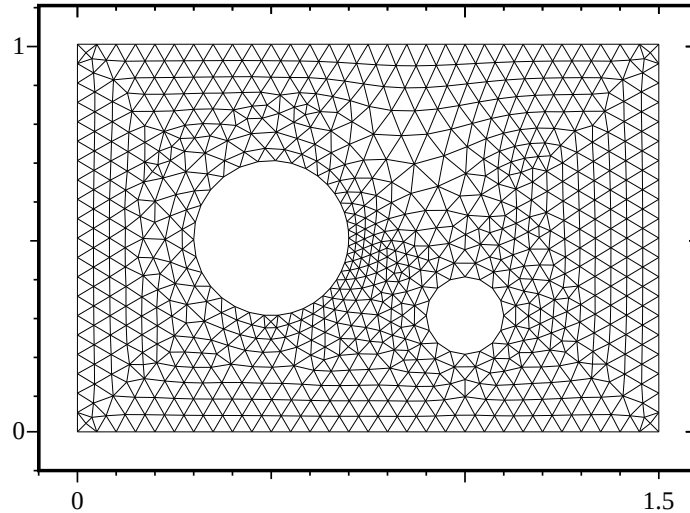


Figure 1: Typical mesh produced by UMesh^{2D}.

2 Overview of the program

In this section the main features of the UMesh^{2D} program are described to provide an overview of the mesh generation process.

2.1 Input files

The program UMesh^{2D} can be run from any directory by issuing the command `umesh2d.exe`. During its execution, the program asks for the name of the problem, for instance, `grid_name`, and assumes that the two files `boundary.grid_name` and `domain.grid_name` exist, prepared according to the correct format (see section 3). The extension `.grid_name` indicates the name chosen by the user to identify his grid together with all the associated files. The two input files contain the data defining the domain to be triangulated and the characteristics of the mesh that the user wants to produce.

The file `boundary.grid_name` consists of two parts. The first part contains the data for the geometrical description of the boundary, which in general may be composed of several edges (*i.e.*, portions of the boundary). The second part of the file accounts for the connection between the extremities of the edges, *i.e.*, the *boundary topology*. Moreover, it contains the data for controlling the size of the triangles on the boundary, namely the *boundary metric*.

The file `domain.grid_name` allows the user to control the size of finite elements (triangles) inside the computational domain by specifying their typical dimension in a selected number of internal points. This set of data will be denoted as the *domain metric*.

2.2 Grid generation process

The algorithm for grid generation proceeds basically as follows. A set of points is generated on the boundary with a spacing (local dimension) determined by the boundary metric. These points are then connected to each other and to any additional point (backgrid point, see page 10) specifying the domain metric to form a *background grid*, called also *backgrid* (Fig. 2).

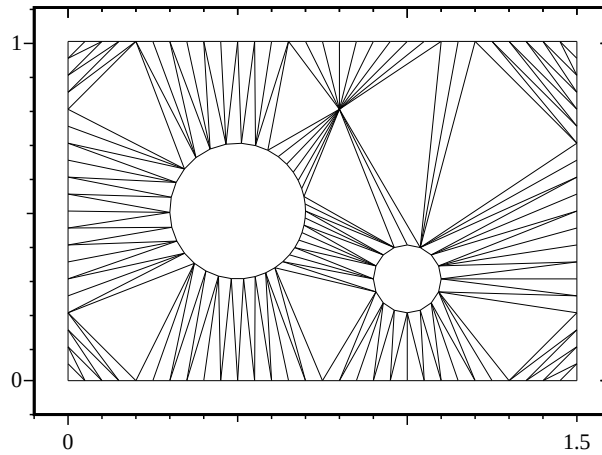


Figure 2: The background grid with one backgrid point inserted.

The background grid is then used to produce on the whole domain the metric information needed by the mesh generation process: basically, the metric assigned on the boundary is interpolated inside all the triangles of the backgrid.

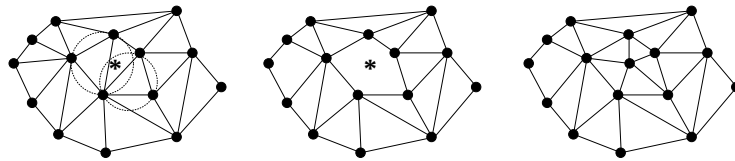


Figure 3: Schematic description of Bowyer–Watson algorithm.

The background grid is taken as the initial Delaunay triangulation to which internal grid points are added, one by one, by means of the Bowyer–Watson algorithm (Fig. 3). New points are chosen by Rebay’s algorithm which is based on an *advancing front* strategy. This method, in its earlier stage, selects one point to be added for every triangle which is adjacent to the boundary and is also of a low quality with respect to the interpolated metric. As new points are added,

the grid is refined and the *front*, that is the frontier between accepted triangles and low quality triangles, moves stepwise towards the interior of domain.

In most cases, the mesh obtained during the generation process is not yet fully acceptable, due to the presence of too stretched triangles. Steiner points, namely fixed points which will belong to the final grid, are possibly added. However, a *Laplacian smoothing* technique based on a spring analogy can be activated by the user to move the points automatically, thus reducing the stretching and simultaneously achieving elements with the requested size.

2.3 Output files

The run of program `UMesh2D` produces six files. Two of them are auxiliary files which summarize informations useful for the user. Other two files collect informations for visualizing the background grid and the generated mesh. The last two files contain all the geometrical and connectivity data of the final mesh.

The two auxiliary files are:

- `knots.grid_name`, which contains the coordinates of the points of each edge of the boundary, specified by the user in the file `boundary.grid_name`,
- `geometry.grid_name`, which contains some data about the interpolant curves associated with each boundary edge.

The files collecting informations for visualization purposes are:

- `backplot.grid_name`, which contains informations about the background grid,
- `gridplot.grid_name`, which contains informations needed to plot the generated grid.

The background and final grids can be viewed by means of the program `Plotmtv` by issuing respectively the commands `plotmtv backplot.grid_name` and `plotmtv gridplot.grid_name` from the directory in which the two files are stored.

All the data which constitute the final generated mesh are contained in the last two output files:

- `nodes.grid_name`, which contains the Cartesian coordinates of the nodes of the mesh,
- `grid.grid_name`, which contains the connectivity matrices for the element-to-node correspondence: connectivity matrices of domain elements and of boundary elements are provided separately.

3 Input file for `UMesh2D`

To better explain the grid generation process we will refer to an example. We will describe how to produce the grid shown in figure 1. Figures and tables in the next sections refer to this example. The tables contain the input data, or part of thereof.

To produce a grid, the user must provide the two files `boundary.grid_name` and `domain.grid_name` written according to the correct syntax, as illustrated in Tables 1 and 3.

3.1 The file `boundary.grid_name`

Input of the global data of the boundary. The first information that must be provided to the mesh generator in the file `boundary.grid_name` is a record for selecting between a grid suitable for Finite Element computation and a grid for Finite Volume computation. In the latter case, `UMesh2D` produces a grid with duplicated boundary nodes for the (very complex) treatment of the boundary conditions in hyperbolic problems. The first record of `boundary.grid_name` must contain two characters, either “FE” or “FV”, to select one of the two possible aforementioned choices.

The boundary consists of different, possibly curved, parts linked to each other. Each distinct part is called a boundary *edge*. The boundary consists of a number `ne` of edges and a number `nv` of vertices. The first two (integer) data of the file `boundary.grid_name` are the number of vertices `nv` and of edges `ne`:

`nv` total number of boundary vertices

`ne` total number of boundary edges

The input line containing the two input data must be preceded by a *control record* with symbols `nv` and `ne` written explicitly. Similar *control records* must be written before other lines containing input data, as shown in Table 1. A free format is allowed in the input lines, *i.e.*, one blank character is equivalent to any number of consecutive blank characters. Records beginning with the character `#` contain comments.

```

#
##### CHOICE BETWEEN FINITE ELEMENT vs VOLUME BOUNDARY NODES #####
#
FE
#
##### GLOBAL DATA OF THE BOUNDARY GEOMETRY #####
#
nv      ne
  6      6
#
##### DESCRIPTION OF EACH BOUNDARY PART #####
#
BEGIN 1 (left side)
line
61  0.0 0.0  0.0 1.0
END 1
#
BEGIN 2 (top side)
line
61  0.0 1.0  1.5 1.0
END 2
#
.....
#
BEGIN 5 (larger circle)
circle
100  0.5 0.5  .2  0 360
END 5
#
BEGIN 6 (smaller circle)
data
small_circle_file_name
translation
  1.0 0.3
rotation
  0.0
END 6
#
##### TOPOLOGY AND METRIC #####
#
BEGIN 1
      e      n      bv      ev
      1      2      1      2
      i      s      h
      1      0.0  0.05
      1      1.0  0.05
END 1
#
BEGIN 2
      e      n      bv      ev
      2      2      2      3
      i      s      h
      1      0.0  0.05
      1      1.0  0.05
END 2
#
.....
#
BEGIN 5 (larger circle)
      e      n      bv      ev
      5      4      5      5
      i      s      h
      1      0.0  0.02
      1      0.3  0.05
      1      0.7  0.04
      1      1.0  0.02
END 5
#
BEGIN 6 (smaller circle)
      e      n      bv      ev
      6      2      6      6
      i      s      h
      1      0.0  0.04
      1      1.0  0.04
END 6

```

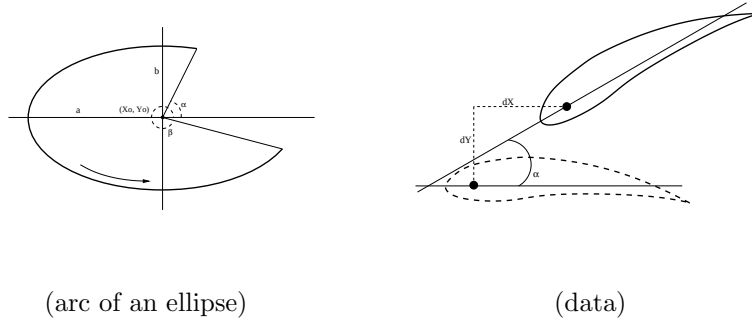
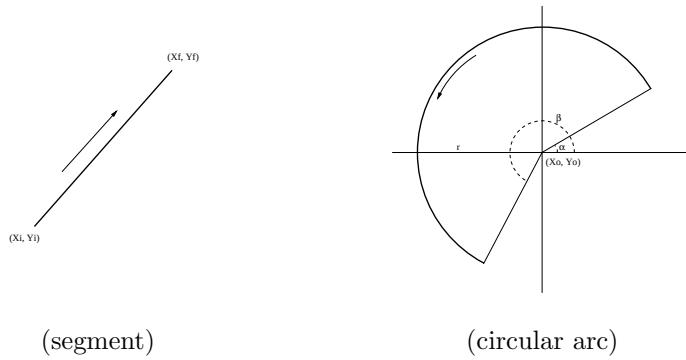
Table 1: The `boundary.two_cicles` file.

Revised by N. CALOSSO and L. QUARTAPELLE on October 15, 2013

Input of the geometry of the boundary. Every edge begins at a vertex and ends in a vertex. For a closed boundary, the two vertices belonging to any pair of consecutive edges must be coincident.

In the file `boundary.grid_name`, every set of input lines is regrouped within the BEGIN and END statements, the former possibly containing some comments, written at a complete discretion of the user. Each edge is defined by specifying either its geometrical character together with a set of suitable parameters or, alternatively, the name of a data file which contains the list of the Cartesian coordinates of the points.

The possible geometrical choices for an edge and for the associated parameters are:



- **line**
number of points for spline approximation (which must at least exceeds the number of points where the metric along the boundary is imposed, see below), coordinates of the begin point, coordinates of the end point;
- **ellipse**
number of points for spline approximation, coordinates of the center, length of the x and y semiaxis, begin angle, end angle (degrees);
- **circle**
number of points for spline approximation, coordinates of the center, radius, begin angle, end angle (degrees);
- **data**
name of the file containing the Cartesian coordinates of a set of points;

the first record must contain the number of points and each subsequent record contains the two Cartesian coordinates of the point (see Table 2),

100		
	0.1000000000	0.0000000000
	0.0997986676	0.0063423920
	0.0991954813	0.0126592454
	:	:
	:	:
	0.0991954813	-0.0126592454
	0.0997986676	-0.0063423920
	0.1000000000	-0.0000000000

Table 2: The `small_circle` data file.

translation

Cartesian coordinates of a displacement vector to be applied to all points,

rotation

rotation angle (degrees) around the origin to be applied to all points (after the translation).

The parameters and the points of file `data` must be given respecting the assumed positive orientation of the boundary, as illustrated in Figure 4.

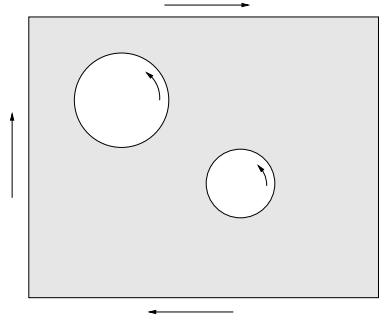


Figure 4: Positive orientation of the boundary.

Input of the topological and metric informations for the boundary.

This part of file `boundary.grid_name` is necessary to connect the edges defined before and to impose the boundary metric along every edge. This is simply obtained by specifying the length of the triangle's base in a few points identified by the values of a normalized arc length parameter (curvilinear abscissa) relative to the edge they belong to (the growth direction of the abscissa is a consequence of the geometry definition). The metric is then interpolated choosing between three options and the program works in such a way so as to spread the metric information inside the domain and to obtain almost equilateral triangles of the proper size everywhere in the mesh.

e edge number

n number of points where the metric is imposed on edge **e** (note that this value must be less than or equal to number of points used for spline approximation on the line, ellipse or circle)

bv begin vertex for edge **e**

ev end vertex for edge **e** (coincident with **bv** for a closed edge)

i type of interpolation for the metric along the normalized curvilinear abscissa of the edge **e**: 1 linear, 2 sinusoidal, 3 geometrical

s increasing curvilinear abscissa, normalized to 1, for imposing the metric

h normalized length of the boundary element

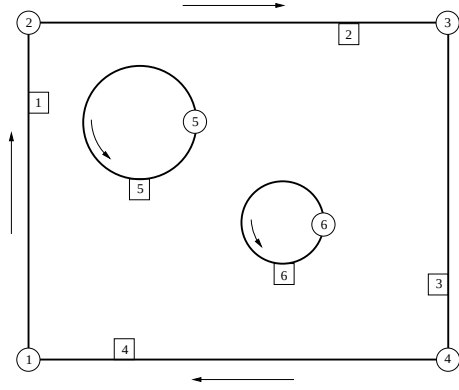


Figure 5: Boundary topology.

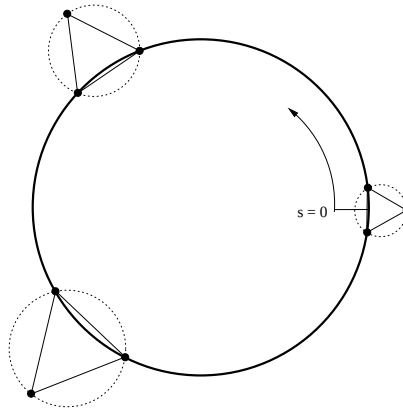


Figure 6: Boundary metric.

For an explanation of the main errors that the user could make, see section 5.2. It is important to remember that no blank record is allowed between two input lines and that the last record of the input must be END.

3.2 The file `domain.grid_name`

The second input file of UMesh^{2D} is `domain.grid_name`. It contains a few additional data to control the size of the triangles *inside* the domain. The first two data must be always provided by the user and are:

nstep is the maximum number of subsequent steps allowed to the program in its generation process of points necessary to improve the grid quality and to match the metric requirements. If the number indicated is not sufficient, a message on the screen will show the percentage of triangles larger than requested by the input.

lsmooth is a “boolean integer” for including (1) or avoiding (0) the Laplacian smoothing.

It is possible to insert a list of points (backgrid points) together with a specified metric and also a list of points (Steiner points) without any explicit metric. In the second case, the metric at the Steiner points is generated automatically by an interpolation on the background triangles they belong to.

Positioning the backgrid points and choosing the values of their corresponding metric is an effective mean for modifying the grid layout, which is otherwise influenced only by the metric on the boundary together with its interpolation. The data for controlling backgrid and Steiner points are specified according the following input:

n number of backgrid (or Steiner) points

x,y Cartesian coordinates of backgrid (or Steiner) points

h metric (except for Steiner)

```

nstep    lsmooth(1/0)
1000     1
#
n        (backgrid points)
1
  x      y      hdata
0.80    0.80    0.10
#
n        (Steiner points)
1
  x      y
0.50    0.30

```

Table 3: The `domain.two_circles` file.

4 Output files of UMesh^{2D}

4.1 Intermediate files: `knots.grid_name`, `geometry.grid_name`

The intermediate file `knots.grid_name` contains the points of every boundary edge:

<code>n_curv</code>	number of edges
<code>dim</code>	number of spatial dimensions (always = 2)
<code>points</code>	number of points
<code>idc</code>	0 for <code>line</code> and <code>data</code> , 2 for <code>ellipse</code> and <code>circle</code>
<code>x, y</code>	point coordinates

The intermediate file `geometry.grid_name` contains the data of the interpolant curve associated with the boundary edge:

<code>n_curv</code>	number of edges
<code>nd</code>	number of spatial dimensions (always = 2)
<code>ns</code>	number of curve points
<code>s</code>	normalized curvilinear abscissa
<code>x1, x2</code>	x and y point coordinates
<code>xs1, xs2</code>	x and y derivatives with respect to <code>s</code>

4.2 Files of the final mesh: `nodes.grid_name`, `grid.grid_name`

The file `nodes.grid_name` contains data of the domain and boundary points:

<code>nd</code>	number of spatial dimensions (always = 2)
<code>np_D</code>	number of points belonging to the domain
<code>np_B</code>	number of points belonging to the boundary
<code>j</code>	domain point index ($1 \leq j \leq np_D$)
<code>rr(1:2,j)</code>	Cartesian coordinates of each point
<code>jB</code>	boundary point index ($1 \leq jB \leq np_B$)
<code>jD_jB(jB)</code>	domain point index corresponding to boundary point index
<code>integer</code>	irrelevant for FEM (FE)

The file `grid.grid_name` contains data of the domain and boundary elements:

<code>ne_D</code>	number of domain elements
<code>ne_B</code>	number of boundary elements
<code>m</code>	index of domain element ($1 \leq m \leq ne_D$)
<code>type</code>	element type, 2 for triangle
<code>j_m(1:3,m)</code>	connectivity matrix, nodes $\leftarrow$ element
<code>ma_m(1:3,m)</code>	connectivity matrix, adjacent elements $\leftarrow$ element
<code>mB</code>	global index of boundary element ($1 \leq mB \leq ne_B$)
<code>type(mB)</code>	element type, 1 for segment
<code>side(mB)</code>	index of the side the boundary element belongs to
<code>jB_mB(1:2,mB)</code>	boundary connectivity matrix, boundary nodes $\leftarrow$ boundary element
<code>mBa_mB(1:2,mB)</code>	boundary connectivity matrix, adjacent boundary elements $\leftarrow$ boundary element

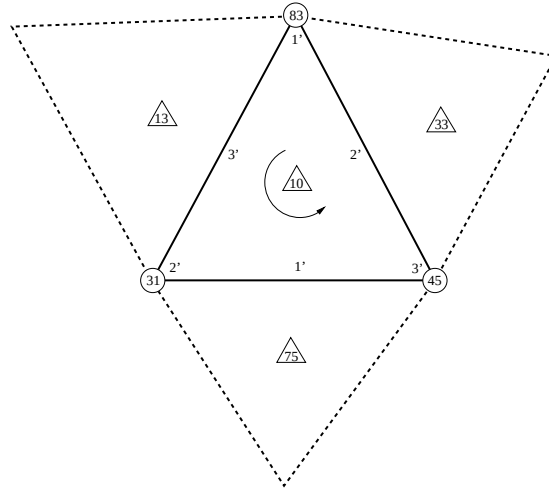


Figure 7: Global and local indices (see table 5).

```
#####
# NAME: two_circles #
#####
# nd np_D np_B #
# 2 915 304 #
#####
# ***** DOMAIN ***** #
#####
# ++++ NODES ++++ #
# j #
# rr(1:2, j) #
# 1 #
# 0.218614037E-01 0.218693044E-01 #
# 2 #
# 0.147814262E+01 0.218775883E-01 #
# 3 #
# 0.147810419E+01 0.978162969E+00 #
# 4 #
# 0.218773470E-01 0.978142850E+00 #
# 5 #
# 0.878989450E+00 0.687758689E+00 #
# ..... #
# ..... #
# 914 #
# 0.107071034E+01 0.229289218E+00 #
# 915 #
# 0.109239125E+01 0.261730356E+00 #
#####
# ***** BOUNDARY ***** #
#####
# ++++ NODES ++++ #
# jB jD_jB(jB) integer #
# 1 764 1 #
# 2 770 1 #
# 3 770 1 #
# 4 771 1 #
# 5 771 1 #
# 6 772 1 #
# ..... #
# ..... #
# 303 915 6 #
# 304 769 6 #
```

Table 4: The `nodes.two_circles` file.

4.3 Plotting files: `gridplot.grid_name`, `backplot.grid_name`

The file `gridplot.grid_name` contains the data to plot the domain grid, while the file `backplot.grid_name` contains the data to plot the background grid.

These two files are saved by the program `UMesh`^{2D} in `plotmtv` format. The background and final grids can be viewed by means of the program `Plotmtv` by issuing respectively the commands

```
plotmtv gridplot.grid_name and
```

```
plotmtv backplot.grid_name
```

from the directory in which the two files are stored.

```

#####
# NAME: two_circles #
#####
# ne_D ne_B #
# 1680 152 #
#####
# ***** DOMAIN ***** #
#####
# ++++ ELEMENTS ++++ #
# m type #
# j_m(1:3, m) #
# ma_m(1:3, m) #
# 1 2 #
# 764 1 770 #
# 153 0 100 #
# 2 2 #
# 770 646 771 #
# 154 0 155 #
# 3 2 #
# 771 644 772 #
# 156 0 154 #
# .... #
# .... #
# .... #
# 10 2 #
# 83 31 45 #
# 75 33 13 #
# .... #
# .... #
# .... #
# 1679 2 #
# 6 17 5 #
# 1678 1672 1680 #
# 1680 2 #
# 7 17 6 #
# 1679 1674 1677 #
#####
# ***** BOUNDARY ***** #
#####
# ++++ ELEMENTS ++++ #
# mB type(mB) side(mB) #
# jB_mB(1:2, mB) #
# mBa_mB(1:2, mB) #
# 1 1 1 #
# 1 2 #
# 2 0 #
# 2 1 1 #
# 3 4 #
# 3 1 #
# 3 1 1 #
# 5 6 #
# .... #
# .... #
# .... #
# 151 1 6 #
# 301 302 #
# 152 150 #
# 152 1 6 #
# 303 304 #
# 0 151 #

```

Table 5: The `grid.two_circles` file.

5 Messages and errors on using UMesh^{2D}

5.1 Echoes on the screen

For every boundary edge the program shows the number of points resulting from the imposition of the metric. For the background grid it shows the number of boundary points and segments, included in the following number of domain points and triangles.

A part of the displayed output pertains to the advancing front:

<code>step</code>	step number
<code>nfro</code>	number of proposed new points
<code>nadd</code>	number of added new points
<code>max_length</code>	the maximum side length between domain elements

In case the value of `nstep` chosen is not enough to complete the mesh, the program shows the percentage of triangles that do not satisfy the metric specified.

If the Laplacian smoothing is present, the number of iterations and the residual are shown.

The number of domain triangles is written after the label `int_head`.

5.2 Frequent errors

When UMesh^{2D} encounters an error in one of the input files, it alerts the user by issuing some error messages. The list of possible error messages with their respective possible cause is given in the following Tables. Messages of errors in the input file `boundary.grid_name` are listed in the first Table while those in the input file `domain.grid_name` are listed in the second Table.

boundary.grid_name		
Error messages	Possible cause of error	Possible solution
Segmentation fault	wrong number of vertices	check the number of vertices
Loading topology ...Invalid input	wrong number of edges or wrong number of edges and number of vertices	check the number of edges and or number of vertices
command not known	invalid geometry option: only <i>line</i> , <i>circle</i> , <i>ellipse</i> and <i>data</i> allowed	watch out for blank line!
After number of segments command not known	blank line after parameter's description	watch out for blank line
simplex center warning	error in topology section	check in topology
stitch points simplex center warning	intersection between edges	plot your boundary and see ...
Problem between element n and m	the boundary is not closed	move the position of some vertices!
$nv \neq ne$		

domain.grid_name		
Error messages	Possible cause of error	Possible Solution
Backgrid points...Invalid Input for integer editing	wrong number of points	check the file
Background grid...Invalid Input for integer editing	blank line(s)	check for blank line(s)

References

- [1] S. REBAY, Efficient unstructured mesh generation by means of Delaunay triangulation and Bowyer–Watson algorithm, *J. Comput. Phys.*, **106**, 125–138, 1993.
- [2] P.L. GEORGE and H. BOROUCAKI, Delaunay triangulation and meshing: application to finite elements, *Hermes*, 413, 1998.